Automatic Generation of Test Oracles - From Pilot Studies to Application

Martin S. Feather
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA
+1 818 354 1194
Martin.S.Feather@Jpl.Nasa.Gov

Ben Smith
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA
+1 818 353 5371
Ben.D.Smith@Jpl.Nasa.Gov

ABSTRACT
Cost, performance and functionality concerns are driving a trend towards use of self-sufficient autonomous systems in place of human-controlled mechanisms. Verification and validation (V&V) of such systems is particularly crucial given that they will operate for long periods with little or no human supervision. Furthermore, V&V must itself be done at low cost, rapidly and effectively, even as the systems to which it is applied grow in complexity and sophistication.

In response to this challenge we have sought to insert more automation into the V&V process. Through a pair of pilot studies we ascertained the opportunities for, and suitability of, automating various analyses whose results would contribute to V&V. These studies culminated in the development of an automatic generator of automated test oracles. This has been applied to aid testing an AI planning system that is a key component of an autonomous space probe.

The pilot studies and development were organized as loosely coupled joint efforts involving a V&V expert and planner experts. This arrangement made good use of the planner experts' limited time.

Keywords
Test Oracles, Verification and Validation, Analysis, Planning, NASA

1 INTRODUCTION

Cost, performance and functionality concerns are driving a move towards use of self-sufficient autonomous systems in place of human-controlled mechanisms. Space probes exemplify the motivations for this trend. The post-launch operational costs of predominantly human-controlled space probes are considerable, requiring frequent involvement of human operators to monitor their status, and to prepare and transmit detailed sequences of commands to direct their actions. The probes may operate at distances from Earth so great as cause significant communication delays (e.g., Jupiter can be as far away as to require 2 hours round-trip communication time). This leads to complicated command sequences because they must be constructed to anticipate many possible eventualities. It also leads to inflexible behavior by the probes which, uponencountering some unanticipated circumstance, must essentially stop what they are doing and call back home for help. Increasing the on-board autonomy of space probes holds the alluring promise of decreased overall mission cost, and increased performance, functionality and robustness.

However, autonomous systems, by their very nature, will be expected to operate for long periods of time with little or no human supervision. Also, the software components of such systems are complicated. Autonomous components can exhibit a much wider range of behaviorsthan the command sequence execution mechanisms of more traditional designs. Furthermore, they must respond correctly to a wide range of environmental circumstances. Together, these make verification and validation (V&V) of the correctness of autonomous systems a crucial and challenging endeavor.

One response to this challenge is to judiciously insert automation into the V&V process. This must yield benefits (notably, by saving some manual effort and/or achieving some analysis that would be impractical to perform manually). Furthermore, the development of the automation must itself be a relatively rapid and inexpensive process, so that the overall effort yields a net saving in time and cost.

This is the approach we followed in the context of V&V of a spacecraft's autonomous system. The dominant limiting factor we faced was the time of the planner experts themselves. As will be seen, this had ramifications

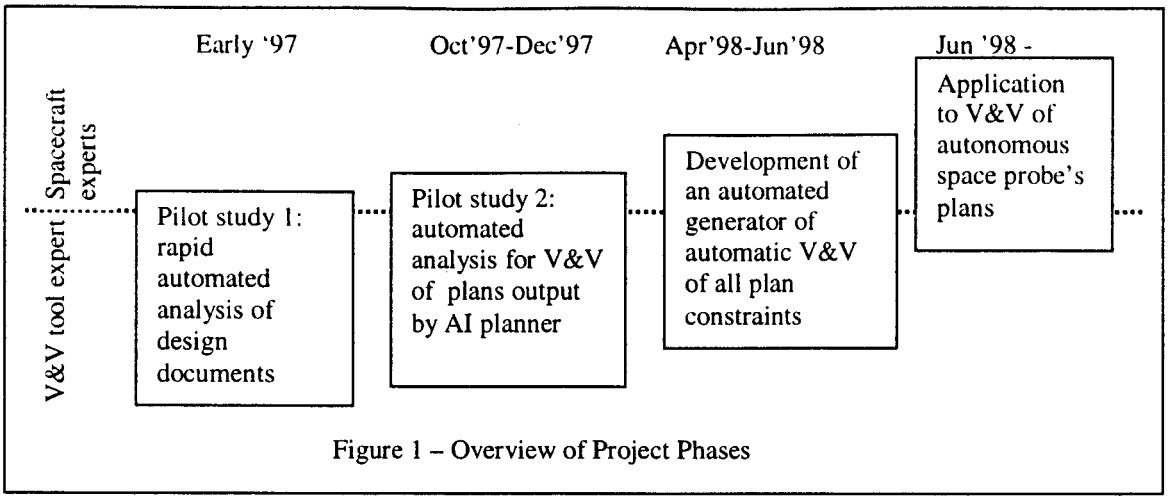


Figure 1 – Overview of Project Phases

throughout. Our overall effort is presented in Figure 1. The four boxes show its major phases; the sections that follow present further details on each of these. The boxes are positioned horizontally to show chronological order, progressing from earliest (at the left) to latest (at the right). The boxes are positioned vertically with respect to a dividing line to show the relative degree of time and effort expended by two classes of personnel. These were the spacecraft personnel, who understood the space probe and its autonomous control system, and the V&V tool person, who understood analysis tools and their construction. Of necessity, we followed a cooperative approach, since neither class of people had the time to become expert in the other area. Indeed, the most critical limiting resource throughout was the time of the spacecraft personnel.

2 FIRST PILOT STUDY: RAPID AUTOMATED ANALYSIS

NASA provides internal research funding to support pilot studies. As the name suggests, a "pilot study" is intended to be a preliminary investigation of some tool, technique or method, applied to a representative problem. A pilot study can thus address somewhat speculative approaches or applications. In contrast, the commitment to rely upon an approach in the actual development of a spacecraft does not have the luxury of such uncertainty. The discovery partway through development that a relied-upon technique, say, is not applicable could have serious repercussions on schedule, budget and even mission success. Pilot studies in NASA (and other organizations) thus play an important role as the first step towards technology transfer from research to actual practice.

The work reported in this paper began with a pilot study, conducted in 1977. The goal of the study was to rapidly develop and apply automated analysis to verify simple properties of spacecraft designs. It was an objective of the study that the approach work with existing design documents, rather than require their manual re-expression into some other notation in preparation for analysis. Another objective was that the analysis itself be rapid and flexible, so that it would yield results quickly, and be amenable to tuning as the analyst gained understanding of the objects of analysis. The approach was founded upon the use of a database as the underlying reasoning engine – see Figure 2. Its four main steps were:

1. Creation of a database schema to represent the design information. This was a purely manual step. Most of the effort went into understanding the design documentation. The actual creation of the schema was

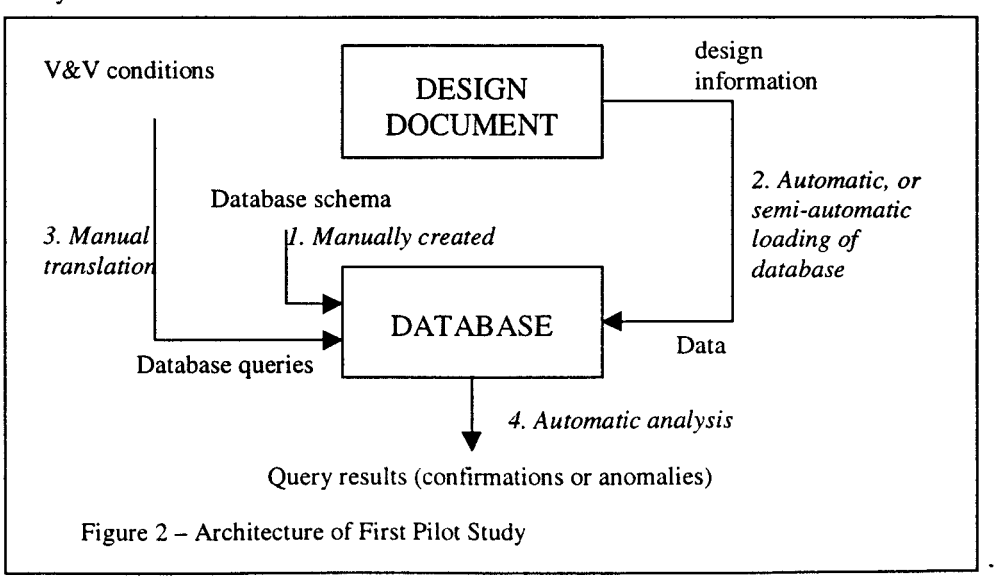


Figure 2 – Architecture of First Pilot Study

straightforward.

2. Loading the design information into the database. This was made a predominantly automated operation, by constructing special-purpose programs to extract information from design documents and translate into the format of the database schema. The high degree of automation made the approach practical for handling voluminous amounts of design information.

3. Determining V&V conditions and expressing them as database queries. Like step 1, the bulk of the effort here went into understanding the domain so as to decide what V&V conditions to analyze for. Having made this decision, it was relatively straightforward to express them as database queries.

4. Analysis. The analysis step was performed by evaluating the database queries against the data, i.e., evaluating the V&V conditions on the design information. The database query mechanism made this step fast and automatic. The reporting of the query results was organized into confirmations and anomaly reports. Generally, these were easy to understand and interpret by the V&V expert and also by the planner experts.

The pilot study examined two sets of design documents – interface diagrams (i.e., summaries of incoming and outgoing connections of software modules) and test logs (i.e., traces of behaviors generated in testing of the software components in simulations). Modest verification conditions were successfully analyzed in this manner, suggesting the viability of the overall approach – see [Feather1998] for details.

This study was conducted primarily by the V&V tool expert. Relatively little time of planner experts was required; their involvement took the form of answering questions and providing clarifications about the design documents, and reviewing the analysis results to confirm their validity (or correcting the V&V tool expert's misunderstandings!).

The study did not focus on the autonomy-specific aspects of the space probe software. It was felt that another pilot study was needed to determine that applicability.

3 SECOND PILOT STUDY: V&V OF AN AUTONOMOUS PLANNER'S OUTPUT

There was concern that autonomous spacecraft control systems might pose a new set of challenges for V&V. The rapid analysis approach of the first pilot study was identified as having potential application to this challenge. In particular, V&V of the AI planning component of an autonomous spacecraft design was to be the focus of the second study. This section provides some background on the autonomous spacecraft, and summarizes this second pilot study.

An Autonomous Space Probe

NASA's "New Millennium" series of space probes is intended to evaluate promising new technologies and instruments. The first of these, "Deep Space 1" (DS1) [DS1 1998], is to be launched in 1998. Increased autonomy is one of several innovative goals that DS-1 will demonstrate [RAX 1998]. The "Remote Agent" [Pell 1996, Pell 1997] will be the first artificial intelligence-based autonomy architecture to reside in the flight processor of a spacecraft and control it for 6 days without ground intervention. The Remote Agent achieves its high level of autonomy by using an architecture with three key modules:

- an integrated planning and scheduling system that generates sequences of actions (plans) from high-level goals,
- a intelligent executive that carries out those actions and can respond to execution time anomalies, and
- a model-based identification and recovery system that identifies faults and suggests repair strategies.

DS-1's planner generates sequences of activities (e.g., turn on camera; take photograph; transfer image to file) that control the various devices of the spacecraft over a period of several days. A series of plans will be generated and executed to conduct the autonomy experiment for a week's duration. The real time executive and diagnosis engine carry out the planned activities in real time, monitoring for, and reacting accordingly to, hardware faults and other surprises if and when they arise. In the case that a hardware fault is disruptive enough to render the current plan unachievable, the real-time executive is able to recognize this, place the probe into a safe state, and invoke the planner again, with an updated state of health of the probe.

The planner is a critical component of the autonomy architecture. Without the ability to generate plans on-board, the spacecraft command sequences would have to be constructed manually, on Earth, and transmitted to the space probe. To guard against this, the planner experts test the planner extensively, in a variety of scenarios. These include scenarios of nominal operation (when the spacecraft hardware is operating as expected) and off-nominal operation (when the spacecraft hardware is malfunctioning). The *correct* operation of the planner in all circumstances is crucial. The command sequences generated by the planner direct navigation, attitude control, power allocation, communication with earth, etc. The entire mission could be jeopardized by an error in an command sequence pertaining to any of these areas. For example, the June 1998 loss of contact with the Solar and Heliospheric Observatory (SOHO) spacecraft is believed to have involved "errors in preprogrammed command sequences" [SOHO 1998].

Automated Verification of Plans' Temporal Constraints

The second pilot study focussed on DS-1's planner. The

objective of this study was to determine the viability of the database-based-analysis approach to verification of key properties of the planner's outputs.

The planner takes as input a description of the initial state, a mission profile (i.e., goals to achieve), and a set of temporal constraints that must hold of the generated plan. The output, a plan, is a schedule of activities for the space probe's hardware and instruments.

The approach pioneered in the first pilot study was applied to DS-1's planner. Figure 3 shows the architecture of the result.

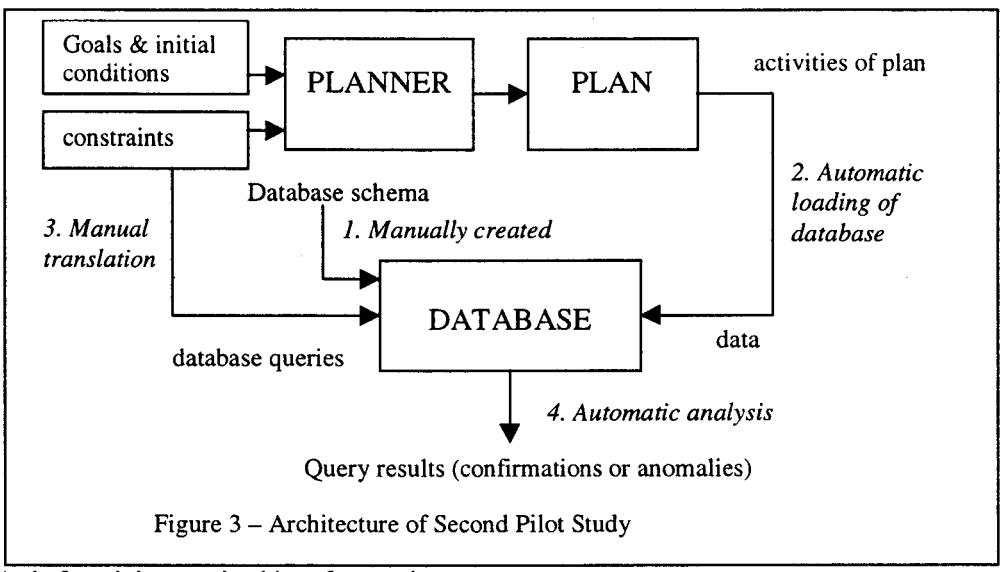


Figure 3 – Architecture of Second Pilot Study

As before, it is organized into four main stages:

1. Creation of database schema to represent the plan's activities. This was a straightforward, manual task. The choice of what information to analyze was determined at the outset of this study. Namely, the activities scheduled by the planner, as appearing in the plans it generates.

2. Loading the database with plan activities. This was made a completely automatic step in this pilot study. The amount of effort to do this was small, in part because DS-1's autonomous agent software and the database used for analysis both happen to be implemented in the same programming language (Common Lisp). The plans generated by the planner take the form of Lisp data structures. It was easy to load such structures into the database – simply provide the data structure definitions, and invoke the Lisp reader! Had there not been this fortuitous coincidence of a common implementation language, it would have been necessary to develop code to parse and translate between linguistic forms. This would have been a standard software programming task.

3. Translation of constraints. For this study, the V&V conditions for analysis were to be chosen from the constraints that served as input to the planner. A representative sample of these was selected for hand-translation into the equivalent database queries. The hypothesis was that this would be straightforward, based on the following reasoning.

   The planner has to be able to generate plans; its constraint language is crafted to simultaneously ease the expression of certain constraints, and limit the form of expression to those that it can readily handle. Conversely, the database only has to be able to evaluate queries about a specific set of data, a far easier task than the search-intensive task of planning. The database query language is an extensible, general-purpose language and so should be capable of straightforwardly expressing the planner's constraints.

   Experience confirmed that it was indeed easy to extend the database language to accommodate the planner-specific constructs. Some translation between syntactic forms was required, but this was readily accomplished by straightforward programming. Examples are given later in this section.

4. Analysis. As before, analysis was automatic, yielding reports of confirmations and anomalies.

The system constructed in this second pilot study was successful in automatically analyzing plans for adherence to a representative sample of temporal constraints. The plans upon which this was tested were actual plans produced during exercising of the DS-1 planner by the planner experts. Such plans were quite verbose – ranging from 1,000 lines to 5,000 lines in length. It was anticipated that thousands of plans would be generated through the course of development. Clearly, the automated checking of constraints would replace an otherwise onerous manual task, and make feasible the thorough checking of all plans.

Pertinent Details of the Second Pilot Study

In this second pilot study, while the amount of time expended by the planner experts remained well below that expended by the V&V tool expert, it was noticeably higher than had the case for the first pilot study. Generally, we attribute this to the need to delve into more application-specific details, resulting in the need for more coaching of the V&V tool expert by the spacecraft planner experts. To convey a feel for this issue, here is an example of one of the

simpler plan constraints expressed in the planner's special purpose language:

```
(Define_Compatibility
  ;;
  ;; Idle_Segment
  ;;
  (SINGLE ((SEP_Schedule SEP_Schedule_SV))
          (Idle_Segment))
    :duration_bounds [1 _plus_infinity_]
  :compatibility_spec
  (AND
   ;; Thrust and Idle segments must all
meet--no gaps
    (meets
     (SINGLE
       ((SEP_Schedule SEP_Schedule_SV))
       ((Thrust_Segment (?_any_value_
                         ?_any_value_)))))
    (met_by
     (SINGLE
       ((SEP_Schedule SEP_Schedule_SV))
       ((Thrust_Segment (?_any_value_
                         ?_any_value_)))))))
```

This simple example illustrates several areas where knowledge held by the planner experts had to be transferred to the V&V expert:

- **Overall application domain:** In the above, "SEP" is an acronym for "Solar Electric Propulsion," the innovative engine that provides thrust to the DS-1 probe. "Thrust" and "Idle" are the two main states this engine can be in.

  Knowledge such as this of the space probe domain provided useful intuition to the V&V expert, and this second pilot study warranted a deeper level of understanding than had been necessary for the first pilot study.

- **Problem-specific terminology:** In the above, "SINGLE" has a connotation specific to DS-1's planner. It introduces a description that matches a single interval. (Alternatives are "MULTIPLE," introducing a description that matches a contiguous sequence of intervals, and "DELTA MULTIPLE" introducing a description that matches a contiguous sequence of intervals that manipulate some quantitative resource, e.g., power). The DS-1 planner experts were the only people who had a complete understanding of this level of detail.

- **Terminological variants:** The overall definition is of a "compatibility", a concept that the V&V expert preferred to think of as a "constraint," in keeping with the terminology of the database tool he employed. Another example is the "?_any_value" term, which serves as a wildcard, indicating any acceptable parameter value may occur in the corresponding parameter position. Again, the V&V expert had the exact same concept, but preferred a different syntax.

- **Confirmation of shared understanding:** there were some areas of shared understanding, but these had to be confirmed, not taken for granted. A trivial example is "AND", which in the above is used to indicate that the constraint [compatibility] holds if all of the clauses of this AND hold. More interesting are the terms "meets" and "met-by," which are binary temporal relations between intervals, drawn from the work by Allen [Allen 1983].

Plans exhibit a similar bewildering mix of domain- and planner- specific information. A small fragment of a plan is shown below. For example, "TOKEN" is a planner-specific term corresponding to an interval.

```
#S(SV-TIMELINE
    :NAME (SEP SEP_SV)
    :NEW-B-TOKENS
      (#S(C-TOKEN
           :CARDINALITY :SINGLE
           :NAME VAL-973
           :SV-SPEC (SEP SEP_SV)
           :TYPE-SPEC ((SEP_STANDBY (0 *
0)))
           :START-B-TOKEN VAL-973
           :END-B-TOKEN VAL-973
           :STATE-VARIABLE (SEP SEP_SV)
           :TOKEN-TYPE ((SEP_STANDBY
                         (0 (:BOUNDS 37800
500000000) 0)))
           :DURATION (37800 500000000)
           :START-TIME-POINT TP-1245
           :END-TIME-POINT TP-1185
           :COMPAT-CONSTRAINTS
            ((ABSOLUTE-START-CONSTRAINT
409277200 909315000)
             (ABSOLUTE-END-CONSTRAINT
909277200 1409277200)
             (ABSOLUTE-END-CONSTRAINT
909277200 1409277200)
             ((CONTAINS 0 500000000 0
500000000) VAL-1651)
             ((CONTAINS 0 0 0 0) SEQ-2694))
       ))
```

The net result was that the V&V expert required an intensive session of coaching on the meaning of the planner notations (plans and constraint language) at the start of this pilot study, and additional assistance at various points throughout.

Example of Translation from Planner Constraint to Database Query

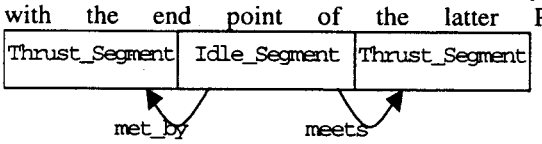
The central question answered by this pilot study was whether the planner constraints could be readily hand translated into the equivalent database queries. The anwer was yes. An illustration follows.

Consider the `Idle_Segment` constraint given earlier. Its

essential core is the following:

```
(SINGLE ((SEP_Schedule ...(Idle_Segment))
:compatibility_spec
(AND
 (meets (SINGLE ((SEP_Schedule ...
                 (Thrust_Segment (?,?)))
 (met_by (SINGLE ((SEP_Schedule ...
                 (Thrust_Segment (?,?))))
```

The fragments `(SINGLE ((SEP_Schedule ...` introduce descriptions that are to match to activities of the SEP scheduled in the plan. The first such description is of an `Idle_Segment` activity. For every instance of an activity in the plan matching that description, the constraint requires that the logical condition `(AND ...)` is true. The logical condition is the conjunct of two clauses. The first says that the matching instance `meets` a `Thrust_Segment` activity, i.e., the end-point of the `Idle_Segment` activity exactly coincides with the start point of some `Thrust_Segment` also in the plan. The second says that the matching instance is `met_by` a `Thrust_Segment` activity, i.e., the start point of the former exactly coincides with the end point of the latter Pictorially,



For translation, this is split into two separate constraints, one for each clause of the conjunct. This allows the checking to be conducted separately for each conjunct, so that any anomaly in a plan can be narrowed down as much as possible. The translated form of the first such conjunct looks close to the following (it has been tidied up slightly for presentation purposes):

```
(A (x) (IMPLIES
    (activity-in-plan x Idle_Segment SINGLE
                                  SEP_Schedule)
    (E (y) (AND (activity-in-plan
         Thrust_Segment SINGLE SEP_Schedule)
              (meets x y)))))
```

`A` and `E` are the database's notations for the logical concepts for-all and exists. `IMPLIES` and `AND` have the standard logical meaning. `activity-in-plan` is a ternary relation (defined for plan checking) that relates an activity name (e.g., `Thrust_Segment`) to a keyword (e.g., `SINGLE`) and schedule (e.g., `SEP_Schedule`). `meets` is a binary relation (again, defined for plan checking) that relates two activities if and only if the end point of the first coincides exactly with the start point of the second.

For this pilot study, a hand-translation was also done for some of the more complex planner constraints. Their additional complexity stemmed from references to activities' parameter values. For example:

```
(Define_Compatibility
;;
;; compats on Max_Thrust_Time
;;
(SINGLE
  ((SEP_Thrust_Timer SEP_Thrust_Timer_SV))
       ((Max_Thrust_Time (100 ?reset))))
:compatibility_spec
(AND
 (ends
  (SINGLE
    ((SEP_Time_Accum SEP_Time_Accum_SV))
         ((Accumulated_Thrust_Time
           (100 0 ?reset
            WHILE_NOT_THRUSTING)))))
 ...))
```

This definition says that every `Max_Thrust_Time` interval whose first parameter is `100` must end an `Accumulated_Thrust_Time` interval whose four parameters are respectively `100`, `0`, the same value as `Max_Thrust_Time` interval's second parameter, and `WHILE_NOT_THRUSTING`. `?reset` is being used as an implicit logical variable that constrains the values in all the places it occurs (namely, the second parameter of `Max_Thrust_Time`, and the third parameter of `Accumulated_Thrust_Time`) to be the same.

Hand-translation of the above would be to:

```
(A (x) (IMPLIES
    (AND
     (activity-in-plan x Max_Thrust_Time
                   SINGLE SEP_Thrust_Timer)
     (parameter x 1 100))
    (E (y)
     (AND
      (activity-in-plan y
         Accumulated_Thrust_Time SINGLE
         SEP_Time_Accum)
      (parameter y 1 100)
      (parameter y 2 0)
      (parameters-equal x 2 y 3)
      (parameter y 4 WHILE_NOT_THRUSTING)
      (meets x y)))))
```

Constraints on activities' parameter values are dealt with by using two more planner-specific relations:

- `parameter`, a ternary relation of an activity, an index into its parameters, and a value. This relation is true if and only if the activity's indexed parameter holds that value.
- `parameters-equal`, a 4-ary relation of an activity, an index into its parameters, another activity, and an index into that second activities parameters. This relation is true if and only if the two activities' indexed parameters hold the same value.

Hand-translations of examples such as these confirmed that DS-1's constraints could be realized as straightforward database queries. On occasion the translated forms were somewhat verbose, as illustrated above in the line-by-line

checks of individual parameter values. Overall, this demonstrated the feasibility of the approach, but suggested that hand-translation of a large number of constraints could be tedious.

4 COMMITMENT TO DEVELOP ANALYSIS TOOL

The success of the second pilot study led to the next phase of the project – a commitment to develop an analysis tool that would be used during testing of the planner by the planner experts themselves. While this might appear to be just a small extension of the previous phase, there were several important ramifications of this transition from pilot study to actual development:

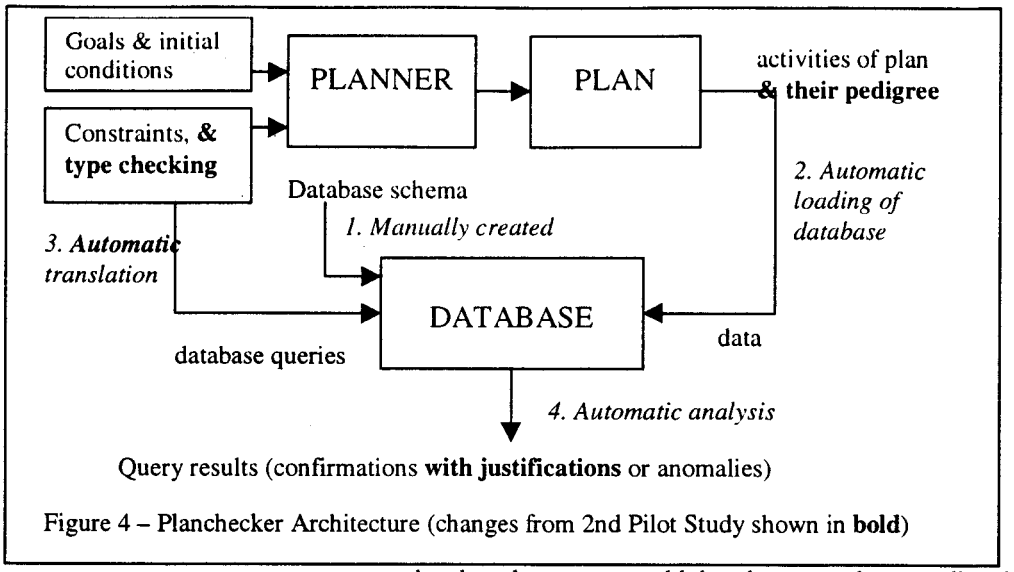


Figure 4 – Planchecker Architecture (changes from 2nd Pilot Study shown in **bold**)

- **Reliance upon the result:** The pilot shadowed the actual space probe development effort, but did not promise to yield results upon which that development effort would rely. Indeed, a valid result of the pilot study could have been that the approach was infeasible. In contrast, this phase committed to the development of a tool that the project would rely upon during testing.

  The positive results of the pilot studies were necessary precursors to this commitment. Additionally, our realization that the analyzer employed an extensible, general-purpose language gave us confidence that it would be capable of handling any combination of constructs that the planner constraint language employed.

- **Developer and end-user different people:** The pilot study tools were developed primarily by the V&V expert, and used by that same person. In contrast, this phase committed to the development of a tool that would be used by the planner experts (notably, the planner experts) with little, if any, involvement of the V&V expert during use.

  This motivated two extensions to the approach demonstrated in the second pilot study: (i) automating the translation from planner constraints into database queries, and (ii) rendering the outputs of the analysis step in terms understandable by the planning experts.

- **End-user agenda:** the DS-1 planner experts constructed an agenda of capabilities they desired of the to-be-developed tool. This featured a prioritized list of capabilities, such that the capabilities to be developed sooner would be the ones they predicted would be of more value to them.

  The preceding pilot studies had helped by providing illustrations of the kinds of analyses that could be accomplished employing this approach. The fact that those illustrations were in terms of DS-1 specific information contributed to their (the planner experts) ability to see its potential. They were thus able to formulate an agenda at this stage, supplanting what was previously the V&V tool expert's *guess* as to what analyses might be interesting and/or valuable.

The architecture of the system developed in this phase is shown in Figure 4. For the remainder of this paper we will refer to this system as the "planchecker". It has the same four stages as the earlier pilot study, but with some additional capabilities, outlined briefly next:

- **Additional analyses:** the planner experts asked for further analyses of the temporal constraints, and analyses of additional information beyond just the temporal constraints themselves. These required loading additional information from plans into the database, and development of additional database queries.

- **Automatic translation:** there were over 200 planner constraints (counting each lowest-level clause as one constraint) that were temporal in nature. Based on the observations of the second pilot study, we recognized that manual translation of the whole set would be a tedious task. This would be even worse if, as was expected, the set of planner constraints would grow and change over time. In keeping with our overall goal of judicious use of automation, it was decided build an automatic translator that would take *any* constraint expressible in the planner language and generate the equivalent database query.

- **Extended output:** the planner experts wanted the query results to report more than simply "OK" when a plan passed the checks. In essence, they wanted a justification for *why* a temporal constraint was satisfied. For example, a constraint that says every SEP-thrusting interval is followed by an SEP-idle interval would be justified by listing, for each SEP-thrusting interval, the specific SEP-idle interval found to satisfy the constraint.
- **Coverage analysis:** the planner experts also wanted to know *which* of the planner constraints had been exercised in the plan. For example, only plans that involved intervals of SEP thrusting would exercise a constraint of the form "every thrusting interval must ...".

Pertinent Details of the Planchecker Development

Like the two pilot studies, the tool was developed primarily by the V&V tool expert, with assistance from the planner experts. This development required a somewhat greater contribution of the planner experts' time, spread throughout the course of the effort.

The hallmark of this effort was the need to deal with many small (and to the V&V tool expert often surprising) details. Most commonly, these were details of the plan constraint language that the V&V tool expert had not encountered earlier. The previous pilot study had employed examples of constraints, drawn from DS-1's entire set of planner constraints. While representative, those examples did not cover the full range of constructs. The discovery of these came to light when the partially developed planchecker was applied to increasingly more of the entire set of DS-1 constraints, and to increasingly many of the plans that had been generated. They manifested themselves in one of three ways:

1. **Error (break) during translation, loading or analysis.** The automatic translator (from planner constraints to database queries) would stop in a run-time break during translation if it encountered unexpected syntax in a constraint. For example, the V&V tool expert assumed length of intervals would be specified in the constraint as integers. When the translator, built to this assumption, encountered a variable name in place of one of these integers, it halted at that spot. Similar breaks could occur during loading (e.g., when an unanticipated use of syntax occurred in a plan) or analysis (e.g., when assumed interrelationships did not hold of the data).

   These details were easy to find and understand. Finding them was simply a matter of applying the translator to a broader set of constraints and plans, and waiting for it to break! Once the V&V expert had his attention drawn to a specific use of syntax in a constraint or plan it was easy for him to recognize the discrepancy. A break in the middle of analysis required some simple debugging-like activity to trace back to the underlying source of misunderstanding. Since the database was implemented on top of Common Lisp, the power run-time environment available in the middle of a break made this task fairly simple.

   All these cases resulted in a simple question that the V&V expert would ask of the spacecraft planning experts (e.g., "what does it mean to use a variable name as a range value where normally there is an explicit integer?")

2. **False alarms - spurious anomalies detected by analysis**. Often the automated steps would complete, but would report a whole host of (as it turned out, spurious) anomalies. The V&V tool expert generally interpreted a large number of anomalies to be indicative of a flaw in his understanding, rather than a grossly incorrect plan. Indeed, genuine plan anomalies were so few and far between that this was an effective working hypothesis.

   The crucial issue in these cases was finding the underlying cause of the spurious anomalies. The V&V expert would spend time to narrow down the likely cause of a reported anomaly. This culminated in a question to ask of the spacecraft planning experts. For example, suppose this was the first analysis of a plan that exercised default interval range values for one of the temporal relationships. An "anomaly" that could be traced back to one of these defaults would be indicative of a misinterpretation of what the default should be. The V&V expert would then know to ask a specific question about that default value.

   This was a somewhat labor-intensive process for the V&V tool expert. Its benefit was that it ensured that the planner experts' (very limited) time was not squandered unnecessarily.

3. **False positives – failure to detect anomalies.** The surprises that were hardest to recognize and understand were those concerning failure to detect anomalies.

   In order to have a known source of these, the V&V tool expert would seed genuine plans with deliberate errors, and observe whether the analysis caught them.

   Another way toward uncovering them stemmed from an addition the planner experts had made to the development agenda. DS-1 generated plans contained activities *and* traceability information on what was taken into account in planning those activities. For example, an instance of the `Thrust_Segment` activity (engine on) in the plan would be accompanied by a trace of the constraint requiring it to be preceded by, and followed by, `Idle_Segment` activities (engine off). The planner experts had asked that the planner

check the validity and completeness of this trace information, as well as checking that the constraints held. The planchecker would report an anomaly if it found a piece of trace information was missing, or if it could not justify the presence of a piece of trace information. The latter especially was helpful toward assuring the completeness and correctness of the planchecker itself.

In more general terms, we were able to take advantage of redundancy within the information being analyzed. This increased our confidence in the validity of the information itself, and in the validity of the tool that checked that information.

Development proceeded iteratively, following the prioritized agenda set by the planner experts. At a finer granularity, the V&V expert went through repeated cycles of discovering and responding to details as they came to light, as discussed above. In retrospect, it is easy to see that following standard software engineering principles and practices could have mitigated some of the problems that arose. For example, the translator from planner constraint language to database query language was coded procedurally, but should have used some grammar-based tool (e.g., POPART [Wile 1997]) permitting a declarative style of specifying translations. This would have facilitated the rapid modification of the translator as new requirements emerged.

The translations themselves (i.e., the database queries and report generation code) became noticeably more complicated than those of the second pilot study. Undoubtedly they could be simplified somewhat given time to reflect and rework (the second-time-around phenomenon). However, some of the complexity reflects a transition from the pilot study's overly simplistic notion of checking a plan. Again, much of the complexity can be ascribed to details specific to the application. Some examples follow:

- All the DS-1 planner constraints take the overall form: for every activity-1 that matches description-1 there exists an activity-2 that matches description-2. A constraint of this form is *trivially* satisfied if the plan contains no activities matching description-1. The planchecker separates such trivial cases in its reports of constraint satisfaction.
- The DS-1 planner generates plans for a segment of the entire mission (e.g., one week). Thus a plan is bounded within some "horizon"– it has a start and an end. Yet, the constraints may extend across this planning horizon. We have already seen the constraint requiring that every Idle_Segment meets a Thrust_Segment. A plan that ends in an Idle_Segment would appear to fail this constraint. In practice, the checking of this constraint must be refined to recognize an Idle_Segment falling at the end of the current plan. Such an instance is reported as a special kind of constraint satisfaction in which the plan satisfies the constraint within its horizon, but defers some residual checking for the next plan. The details of all such deferred checks are included within the planchecker's report.
- A few of the constraints reference information that is not stored in plans. In essence, this external information directs which one of several constraints is to apply. The planchecker's constraint translations handle these circumstances by checking each alternative. If all fail, it is an anomaly. If the plan is found to satisfy one of the alternatives, again, a special kind of constraint satisfaction is reported, which included the deduction of what the external information must be to direct the choice of the satisfied constraint. (Incidentally, this was one of the more challenging examples of "false alarms" that the V&V tool expert encountered. When it arose, he had to approach the planner experts with the plan and the violated constraints, but no working hypothesis of what could account for the violation!)

In summary, the V&V tool expert developed the planchecker tool within the allotted development schedule and fulfilling most of the capabilities asked for in the development agenda. The transfer of the tool to the planner experts, and their subsequent use of it, are described briefly next.

5 USE OF ANALYSIS TOOL

The planchecker was installed on the spacecraft planning experts' system for them to use. During testing they exercise the DS-1 planner on many varied scenarios of operation. Each test (usually) yields a plan, and the planchecker is applied to check each such generated plan. Its results are accumulated along with other statistics about the plan generation, e.g., how long it took to generate the plan, how much memory was required to do so. The planchecker runs automatically, and can be easily applied in "batch mode" to a whole series of plans. It is tolerably efficient, taking on the order of 2 minutes to complete the checking of a typical plan. The key observation is that interaction with the V&V tool expert is *not* required during this process.

Over the course of use, several sets of changes have been made to the planner constraints. Re-translating the entire set of constraints, to generate a new instance of the test oracles, easily accommodated these changes. On these occasions the V&V tool expert has been on hand. They have gone smoothly, with only one instance of the need to step in and make a corrective modification.

The spacecraft planner experts have extended their use of the planchecker in one particularly interesting manner. This

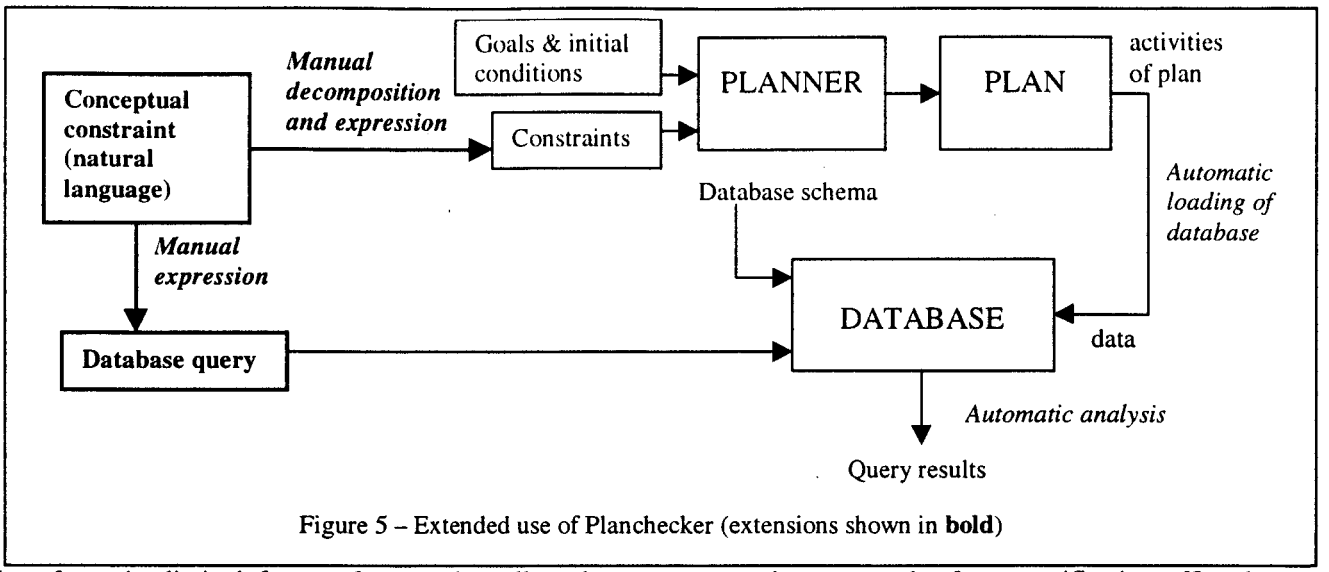


Figure 5 – Extended use of Planchecker (extensions shown in **bold**)

arises from the limited forms of expression allowed as input to the planner. On occasion, they have found it necessary to decompose a fairly obvious constraint that they want the plans to exhibit into a *set* of constraints that the planner will accept, and that in combination will achieve the original constraint. This is a manual process. Because the database query language is not so tightly constrained, it is often possible to hand-express their original constraint into a *single* database query. This can then be applied to automatically check plans. Doing so gives them increased confidence in the validity of their manual translation of the original constraint into multiple planner constraints. Figure 5 shows the architecture of this extended use of the planchecker.

The implications of this are twofold: (1) the planner experts have mastered the use of the database language and the special-purpose constructs added to represent and reason about plans. Seeing examples they were already familiar with, namely translations of the standard constraints, helped them rapidly achieve this level of understanding. (2) Their extensions take advantage of some of the automation of the planchecker architecture – the automatic loading of plans into the database, and the evaluation of database queries. (They cannot, of course, use the translator from planner constraint language, because their original constraints are not expressible in that language.) This has meant they have attained the extra validation at the cost of very little extra time and effort on their part.

6 CONCLUSIONS

This overall task was relatively modest in scale, and involved only a small number of personnel. Hence all of our conclusions should be taken to refer to relatively small-scale software development.

Our work follows the trend towards the use of automation in testing. [Richardson 1994] presents an approach to generating test oracles from specifications; [Jagadeesan et al, 1997] present a feasibility study on automatically testing software for violations of safety properties expressed in temporal logic. Our work culminated in the insertion of a high degree of automation into the generation of the test oracles themselves. (Note that we have not addressed the generation of test cases, which is also an interesting and important problem). Our experience on applying this to a real-world (actually, out-of-this-world) problem suggests that while this may appear a modest objective from a research perspective, putting this into practice can be a non-trivial task.

The lessons we draw from this experience are presented next, beginning with those related to general software engineering principles, followed by those specific to V&V.

Software Engineering Lesson 1: Pilot Studies
Our experience re-iterates several well-understood virtues of pilot studies as a precursor to actual development.

In particular, pilot studies:

- provide evidence of feasibility on specific cases,
- serve as prototypes whose inadequacies point out areas which should be done better the next time around, and
- yield examples which inspire suggestions for extensions, further applications, etc.

In addition to the above, we found it useful to formulate a justification of why the pilot study approach would extend to the full problem. In our case, this was the argument that the planchecker employed a less restricted and readily extended language, and so would be able to accommodate any constraint that the more restricted planner language would likely use. Such a justification nicely complemented the evidence provided by the pilot studies' specific cases.

Software Engineering Lesson 2: "On-Demand" Specification and Development

When domain experts' time is a critical resource, follow an "on-demand" policy of specification and development to make best use of their time and availability.

At the start of the project we lacked a complete and fully documented specification of the task (i.e., plans and the planner language). Furthermore, we needed to make best use of the domain experts' valuable and limited time.

These problems were alleviated by the existence of numerous sample inputs (i.e., plans and planner constraints). Also, the nature of the task clearly circumscribed the areas that the analysis expert would have to master.

We followed an "on demand" approach to knowledge acquisition, where the analysis expert would proceed as far as possible before making the next enquiry of the planner experts. This made good use of the planner experts' limited time and availability, since it kept the sum total of their time small, consumed it in small chunks, and could be done asynchronously (e.g., via email exchanges, supplemented by brief telephone calls).

Software Engineering Lesson 3: Strive for Flexibility

Our overall experience reinforces the value of flexibility, and therefore the value of approaches that yield flexible solutions.

At the time of the first pilot study we did not know that it would lead to the development of a planchecker-like tool, but fortunately our overall approach proved to be sufficiently flexible to accommodate the required extensions and elaborations.

Our major area of dissatisfaction with the planchecker is that its translator component (from planner constraints to database queries) was programmed procedurally. A more declarative style would be superior. In such a style, the translation would be expressed as a set of translation rules, executed by a general-purpose rule engine. For translation tasks of this scale and complexity, the advantages of a declarative style are that the translation rules are readily created, understood and maintained.

In retrospect, such an approach would likely have yielded a net time savings for the V&V tool expert, as well as resulted in a more perspicuous translation tool.

We also speculate that the planner experts, guided by the translations of their planner constraint language, would readily see how to use and write additional translations. Perhaps they could even go on to use the same approach to extend the planner constraint language itself, i.e., to automatically translate the formal expression of a conceptual constraint into the set of simpler constraints that the planner language currently accepts.

V&V Lesson 1: Encourage and Use Redundancy and Rationale

V&V can make good use of redundancy and rationale, to increase assurance in the V&V results, and to assist in the development of the V&V technology itself.

Each plan generated by the spacecraft planner contains both a schedule of activities, and a rationale relating those activities to the constraints taken into account in their planning. Checking both of these might appear redundant – surely what really matters is whether or not a plan satisfies all the constraints. Nevertheless, we found this redundancy to be useful in two ways:

1. The planner experts gained additional assurance that their generated plans were correct, in particular, that they generated the "right" results "for the right reasons."
2. The V&V tool expert made use of the redundancy to extend (and debug) his understanding of the task. Every constraint that the planchecker identified as being involved had to be identified in the plan's rationale, thus forcing the planchecker to be complete and correct in its treatment of rationales. Likewise, every constraint mentioned in the rationale had to be seen to be involved by the planchecker, thus forcing the planchecker to be complete and correct in its treatment of constraints.

V&V Lesson 2: Database-based Analysis

The use of a database as the underlying analysis engine has practical applications and benefits.

Based on the first of our pilot studies we had made the argument that database-based analysis was suited to "lightweight" V&V [Feather1998]. The success of this whole effort strengthens our belief in this position, and highlights some further benefits.

The database approach suggests a natural decomposition of the problem into: translating the V&V conditions into database queries, loading the data into the database, performing the analyses, and generating the reports. This simple architecture nicely separates the key steps. For example, we are about to modify the planchecker's database loading portion in response to a recent change in format of plan structures; this will not require us to modify any of the other steps. Also, this architecture facilitated the planner experts' extended use of the planchecker (i.e., their checking of complex conceptual constraints by manually expressing them as database queries).

The database itself is used as intermediary between analysis and report generation steps. The planchecker places analysis results back into the database, alongside the original data (plans) from which those results are derived. Thus the report generation phase has uniform and simultaneous access to both kinds of data regardless of

source, considerably facilitating the report generation task.

V&V Lesson 3: Analysis Results Need Structure
Test oracles should yield results with far more content and structure than simply "passed" or "failed".

During the pilot studies it had sufficed to yield analysis results with trivial stnpp ructure – they reported that the object had "passed" the analysis test, or had "failed due to...."

The planchecker development entailed the generation of analysis results and reports with considerably more structure to the "passed" cases. For example, reports that identified *which* constraints had been exercised by a plan, and that distinguished *how* constraints had been satisfied: those that were wholly satisfied by the plan, those that deferred some condition to activities beyond the plan's horizons, etc.

We suspect that there may be general principles by which test oracles can be built to yield such structured analysis results, an area we think is worthy of further attention.

ACKNOWLEDGEMENTS

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

REFERENCES

[Allen 1983] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832-843, 1983.

[Cohen 1989] D. Cohen. Compiling Complex Database Transition Triggers. *Proceedings of the ACM SIGMOD International Conference on the Management of Data* (Portland, Oregon, 1989), ACM Press, 225-234.

[DS1 1998] http://nmp.jpl.nasa.gov/ds1/

[Feather1998] M.S. Feather. Rapid Application of Lightweight Formal Methods for Consistency Analyses. To appear in IEEE Transactions on Software Engineering.

[Jagadeesan et al, 1997] L.J. Jagadeesan, A. Proter, C, Puchol, J.C. Ramming & L.G.Votta. Specification-based Testing of Reactive Software: Tools and Experiments. *Proceedings of the 19th International Conference on Software Engineering* (Boston, MA, May 1997), 525-535.

[Pell 1996] B. Pell, D.E. Bernard, S.A. Chien, E. Gat, N. Muscettola, P.P. Nayak, M.D. Wagner & B.C. Williams. A Remote Agent Protoype for Spacecraft Autonomy. *Proceedings of the SPIE conference on Optical Science, Engineering and Instrumentation*, 1996.

[Pell 1997] B. Pell, D.E. Bernard, S.A. Chien, E. Gat, N. Muscettola, P.P. Nayak, M.D. Wagner & B.C. Williams. An Autonomous Spacecraft Agent Prototype. *Proceedings First International Conference on Autonomous Agents*. ACM Press, 1997.

[RAX 1998] http://nmp.jpl.nasa.gov/ds1/tech/autora.html

[Richardson, Aha & O'Malley 1992] D.J. Richardson, S.L. Aha & T.). O'Malley. Specification-based Test Oracles for Reactive Systems. *Proceedings of the 14th International Conference on Software Engineering* (Melbourne, Australia, May 1992), 105-118.

[SOHO 1998] *SOHO Mission Interruption Preliminary Status and Background Report – July 15, 1998* http://umbra.nascom.nasa.gov/soho/prelim_and_background_rept.html

[Wile 1997] D. Wile. Abstract Syntax from Concrete Syntax. *Proceedings of the 19th International Conference on Software Engineering* (Boston, MA, May 1997), 472-480